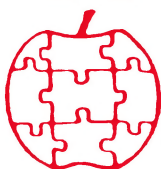


# Apple

\$2.40



# Assembly

# Line

Volume 8 -- Issue 7

April, 1988

Peeking Inside AppleWorks 1.3, Part 5:

Menu Display and Selection . . . . .	2
ProDOS File Transformer (S-C into TXT) . . . . .	18
Modify CATALOG to Show All AuxTypes . . . . .	29
BLOADing ProDOS Directories . . . . .	31
More on trip to Phoenix . . . . .	32

## 68HC11 Cross Assembler Now Here!

At long last, I have written a cross assembler for the Motorola 68HC11. A lot of you have been asking about it, and thanks to the persistence of one customer it is now available. See our ad on page 3 for pricing and a list of other cross assemblers.

## A Visit to Phoenix

The weekend of June 10-12 I attended the AZ Apple Fiesta in Phoenix, Arizona. (How can this be reported in the "April" issue? Sorry, we are still woefully behind in the publishing schedule, it is now late June.) The main attraction for me was a chance to spend a day at Western Design Center with Bill Mensch, the chief designer of the 65816 and its predecessors.

Secondly, I wanted to attend Randy Hyde's mini-conference on a new standard for 6502 family assembly language. Randy was there, with Brian Fitzgerald who now supports and markets the Lisa assembler. Roger Wagner represented the Merlin interests, and Mike Westerfield the ORCA/M and APW. Paul Santa-Maria and Chuck Kelly from ProDev Inc. in Michigan came to see about standardizing a symbol table format for use with symbolic debuggers. I also met P. J. Curran from ProData 21 in Florida. Since one cannot fly the 1000 miles to Phoenix directly from Dallas for less than \$250 each way, I drove 200 miles south to Austin to get a round trip ticket for only \$138. Bill Morgan lives in Austin now, and he decided to come also. Jeff Creamer and Don Lancaster were also involved in this conference. As far as I could tell, we did not make any real decisions about any changes to existing assemblers, or even future ones. Neither did we gain any hard data about the new opcodes and addressing modes of the 65832. But it was a very positive meeting.

(continued on page 32.)

Peeking Inside AppleWorks 1.3, Part 5:  
Menu Display and Selection Subroutines....Bob Sander-Cederlof

Part 5 already? This is rapidly turning into a major series, and a popular one at that. A lot of you have called or written telling how much you like it (thank you!), so I will do my best to keep it interesting and useful.

This month I am going to document some subroutines which constitute a major portion of the "AppleWorks Human Interface". Namely, the menu display and selection subroutines.

We hear so much these days about the MacIntosh human interface, and indeed it has become the standard for the IIgs, like it or not. Personally, I can live with it, but I prefer plain old text menus. Why? They are faster, consume less of the computer's resources (leaving more of them for end users), and are quicker (for me) to use. Well-designed text menus with a consistent screen layout and selection method are easier for me than icons and mice.

How can that be? I thought Apple had proven the icon menu to be easier? I suppose they decided that after asking "the rest of us", leaving out the experienced computer people. But isn't text what we are trained from childhood to read? Some languages are written with icons (Chinese, for example), but MOST languages are written with text. And isn't the text-menu drilled into us from childhood also? Even the word "menu" reminds us of a list of textual lines for selecting the components of a meal. And what about multiple-choice tests?

And even some of the Apple folk believe in text, as is evidenced by the Control Panel and other Classic Desk Accessories on the IIgs.

All the above to justify adding my voice to that of Tom Weishaar. Tom is the author of the Open-Apple newsletter, which all Apple II lovers ought to read. We both think the AppleWorks human interface is a great starting point for the Apple II general operating environment. With just a little improvement, mostly in the area of pathname entry, and just a little generalization, we could use it for running all sorts of applications and compilers for all sorts of languages.

AppleWorks has an amazingly consistent human interface throughout. At almost every level you will see a simple list of numbered choices, with the same control scheme. One of the lines will be displayed in inverse mode, and you can either move the "bar" with the up- and down-arrow keys, or by typing the item number. When the bar is on your choice, the RETURN key will select it. ESCAPE will pop out of the menu to the previous level. You can usually get a help screen by typing Apple-?. The directions for getting help are shown at the right end of the bottom line, and the action ESCAPE will cause are shown at the right end of the top line.

If there happen to be more than nine numbered options, so that some of them have two-digit options, AppleWorks even allows you

S-C Macro Assembler Version 2.0 . . . . . DOS \$100, ProDOS \$100, both for \$120  
Version 2.0 DOS Upgrade Kit for 1.0/1.1/1.2 owners . . . . . \$20  
ProDOS Upgrade Kit for Version 2.0 DOS owners . . . . . \$30  
Cross Assemblers for owners of S-C Macro Assembler . . . . . \$32.50 to \$50 each  
(Available: 6800/1/2, 6301, 6805, 6809, 68HC11, 68000, Z-80, Z-8, 8048, 8051, 8085,  
RCA 1802/4/5, PDP-11, GI1650/70, Mitsubishi 50740, others)  
Source Code of any S-C Macro Assembler or Cross Assembler . . . . . each, additional \$100  
S-C DisAssembler (ProDOS only) . . . . . without source code \$30, with source \$50  
RAK-Ware DISASM (DOS only) . . . . . without source code \$30, with source \$50  
ProVIEW (ProDOS-based disk utility program) . . . . . \$20  
Full Screen Editor for S-C Macro (with complete source code) . . . . . \$49  
S-C Cross Reference Utility . . . . . without source code \$20, with source \$50  
S-C Word Processor, both DOS & ProDOS, both 40- & 80-columns, with complete source code \$50  
DP18 and DPPP, double precision math for Applesoft, including complete source code. . . \$50  
ES-CAPE (Extended S-C Applesoft Program Editor), including Version 2.0 With Source Code . \$50  
ES-CAPE Version 2.0 and Source Code Update (for Registered Owners) . . . . . \$30  
Bag of Tricks 2 (Quality Software) . . . . . (\$49.95) \$45 \*  
S-C Documentor (complete commented source code of Applesoft ROMs) . . . . . \$50  
Copy II Plus (Central Point Software) . . . . . (\$39.95) \$30 \*  
AAL Quarterly Disks . . . . . each \$15, or any four for \$45

(All source code is formatted for S-C Macro Assembler. Other assemblers require some effort to convert file type and edit directives.)

Vinyl disk pages, 6"x8.5", hold two disks each . . . . . 10 for \$6 \*  
Diskette Mailing Protectors (hold 1 or 2 disks) . . . . . 40 cents each  
(Cardboard folders designed to fit 6"X9" Envelopes.) . . . . . or \$25 per 100 \*  
Envelopes for Diskette Mailers . . . . . 6 cents each

Sider 20 Meg Hard Disk, includes controller & software . . . . . (\$695) \$550 +  
Sider 40 Meg Hard Disk, includes controller & software . . . . . (\$995) \$860 +

Minuteman 250 Uninterruptible Power Supply . . . . . (\$359) \$320 +  
Minuteman 300 Uninterruptible Power Supply . . . . . (\$549) \$490 +

65802 Microprocessor, 4 MHz (Western Design Center) . . . . . \$25 \*  
quikLoader EPROM System (SCRG) . . . . . (\$179) \$170 \*  
PROGRAMMER (SCRG) . . . . . (\$149.50) \$140 \*

"Exploring the Apple IIgs" . . . . . Gary B. Little (\$22.95) \$21 \*  
"Apple IIgs Technical Reference" . . . . . Michael Fischer (\$19.95) \$19 \*  
"65816/65802 Assembly Language Programming". . . . . Michael Fischer (\$21.95) \$20 \*  
"Programming the 65816" . . . . . David Eyes & Ron Lichty (\$22.95) \$21 \*  
"Programming the Apple IIgs in C and Assembly Language". . . . . Mark Andrews (\$18.95) \$18 \*  
"Apple //e Reference Manual". . . . . Apple Computer (\$24.95) \$23 \*  
"Apple //c Reference Manual". . . . . Apple Computer (\$24.95) \$23 \*  
"ProDOS-8 Technical Reference Manual". . . . . Apple Computer (\$29.95) \$27 \*  
"ProDOS-16 Technical Reference Manual". . . . . Apple Computer (\$29.95) \$27 \*  
"Apple IIgs Firmware Reference". . . . . Apple Computer (\$24.95) \$23 \*  
"Apple IIgs Hardware Reference". . . . . Apple Computer (\$24.95) \$23 \*  
"Apple IIgs Toolbox Reference, Volume 1". . . . . Apple Computer (\$26.95) \$24 \*  
"Apple IIgs Toolbox Reference, Volume 2". . . . . Apple Computer (\$26.95) \$24 \*  
"Apple IIgs Programmer's Introduction". . . . . Apple Computer (\$32.95) \$30 \*  
"ProDOS Inside and Out" . . . . . Dennis Doms & Tom Weishaar (\$16.95) \$16 \*  
"Beneath Apple ProDOS". . . . . Don Worth & Pieter Lechner (\$19.95) \$18 \*  
"Beneath Apple DOS". . . . . Don Worth & Pieter Lechner (\$19.95) \$18 \*  
"Inside the Apple //c". . . . . Gary B. Little (\$19.95) \$18 \*  
"Inside the Apple //e". . . . . Gary B. Little (\$19.95) \$18 \*  
"Understanding the Apple //e". . . . . Jim Sather (\$24.95) \$23 \*  
"Understanding the Apple II". . . . . Jim Sather (\$22.95) \$21 \*  
"Apple II+/IIe Troubleshooting & Repair Guide". . . . . Brenner (\$19.95) \$18 \*  
"Assembly Language for Applesoft Programmers". . . . . Finley & Myers (\$18.95) \$18 \*  
"Now That You Know Apple Assembly Language". . . . . Jules Gilder (\$19.95) \$18 \*  
"Enhancing Your Apple II, vol. 1". . . . . Don Lancaster (\$15.95) \$15 \*  
"Enhancing Your Apple II, vol. 2". . . . . Don Lancaster (\$17.95) \$17 \*  
"Assembly Cookbook for the Apple II/IIe". . . . . Don Lancaster (\$21.95) \$20 \*  
"Microcomputer Graphics". . . . . Roy E. Myers (\$14.95) \$14 \*

\* These items add \$2 for first item, \$.75 for each additional item for US shipping.  
+ Inquire for shipping cost.

Customers outside USA inquire for postage needed.  
Texas residents please add 8% sales tax to all orders.  
<< Master Card, VISA, Discover and American Express >>

S-C Software Corporation  
2331 Gus Thomasson #125  
DALLAS, TX 75228  
Phone 214-324-2050



to move the bar to the two-digit lines by typing their numbers. When you type a digit, the bar moves to one of the first nine lines. Then if you type a second digit, AppleWorks tries to move to the line with that two-digit number. If there is not one, you will hear a beep. If there is, the bar will move to that line. You can observe this action even on a menu with fewer than ten lines. When you select a line by typing a digit, you will not only see the bar move, but also the help and escape messages will change. On the top line you will see "Escape: Erase entry", and on the bottom line instead of an offer of help you will see "xxxK Avail." (the number of available memory bytes). When you type a second digit, you hear the beep and the help and escape info changes back to what it was before you typed the first digit.

Pretty slick. Until now I always assumed that numeric selection would either be limited to single-digit items, or that I would have to require a terminating character from the user so I would know when he had typed all the digits. Robert Lissner wins again! (Some of you have pointed out his name used to be "Rupert", but I read somewhere that he now prefers "Robert", or Bob.)

AppleWorks also allows the menu items to be anywhere on the screen. They do not even all have to be in one column, or vertically aligned. You can use single, double, or even variable vertical spacing.

Look ahead to the DISPLAY.MENU.LINE subroutine, lines 4580-5190. When you want to put a menu on the screen, you call this subroutine. The variable LAST.LINE.NUMBER.OF.MENU must be initially zeroed, and this subroutine will increment it each time you call. You call this subroutine once for each numbered line of your menu. Call it like this:

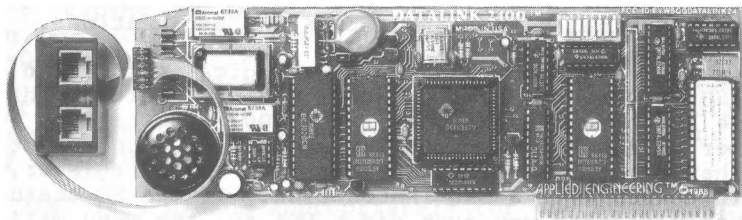
```
JSR DISPLAY.MENU.LINE
.DA #column,#line
.DA string
```

The string parameter is the address of a string which begins with a length byte. Column and line numbers are either absolute or relative. If you give values like 0-23 for line and 0-79 for column, they are absolute. If you add \$80 to line and/or column, it/they become relative to the coordinates stored in MENU.CORNER.LEFT and MENU.CORNER.TOP. Lines 4850-5000 compute the actual coordinates of the beginning of your menu line.

A block of 93 bytes called MENU.TABLE allows for keeping track of 30 menu lines. Each entry in this table is three bytes, and the zero'th entry is not used. Each time you call DISPLAY.MENU.LINE one entry is added to this table. Each three-byte entry consists of the column, line, and length of the menu item.

DISPLAY.MENU.LINE also takes care of prefixing the menu line number. You do not include it in your string. (This allows you to use the same strings in different menus.) Lines

# The new DataLink™ 2400 modem from Applied Engineering, it's a lot more than just twice as fast.



**A**ppled Engineering's new DataLink™ 2400. Simply put, the finest modem on the market for your Apple IIs, IIe or II+.

Bring home a world of information . . . from up to the minute flight information to whole libraries of resource materials. Even download free software and games.

## Twice the speed.

At transmission speeds up to 2400 bps (bits-per-second), Applied Engineering's new DataLink 2400 is capable of putting text on the screen faster than the human eye can follow. That means you can capture a great deal more material in less time than with 1200 bps modems. And *unlike other modems*, the DataLink 2400 comes complete with powerful, easy-to-use communications software.

## Complete communications software included.

Both our new DataLink 2400 and our DataLink 1200 modems feature AE's exclusive communications software — on disk and in ROM—everything needed to get you immediately up and running. Our powerful DataTerm software for the IIs and IIe supports VT-52 screen emulation, macros, file transfers, on-line time display, recording buffer and more. It even stores hundreds of phone numbers for auto-dialing and log on. And for II+ and 64K IIe owners, our OnLine 64 software has many of the same powerful features.

## Worldwide compatibility.

The DataLink 2400 is fully compatible with Bell 103 and 212 protocols, as well as European protocol CCITT V.22 BIS, V.22 and V.21. It operates at varying transmission speeds from 0-300, 1200 and 2400 bps.

The new 2400, like our best-selling DataLink™ 1200, carries a full five year warranty and comes complete with two modular phone jacks for data and voice calls, a thoughtful feature that means fewer wires to connect. We also include an extra long telephone cable, in case your computer is across the room from your telephone jack. You can track the progress of calls either electronically or via on-board speaker. And built-in diagnostics reliably check transmission accuracy.

Prices subject to change without notice. Brands and product names are registered trademarks of their respective holders.

## Packed with important features:

- Non-volatile memory for modem configuration
- Full Hayes AT compatibility
- Point-to-Point, ASCII Express, Access II compatibility, in addition to AE's included DataTerm and OnLine 64 software.
- Super Serial Card "Front End" for highest software compatibility (unlike others)
- Adaptive equalization and descrambling
- Hardware configuration for DSR and DCD
- PC Transporter (MS-DOS) compatibility
- FCC certified design

## \$204.90 in freebies.

We also throw in a nice collection of goodies—a free subscription to the GENIE network worth \$29.95, \$60 of free on-line time from NewsNet, a free \$50 subscription to the Official Airline Guide and a fee-waived membership to The Source worth \$49.95 plus \$15 of free on-line time.

That's \$204.90 worth of free memberships, discounts and

on-line time when you purchase the powerful DataLink 2400 at **\$239**.

## DataLink 1200 reduced.

Loaded with all the features of the new 2400, (except CCITT, DSR/DCD and non-volatile ROM configurations) our 1200 bps DataLink modem, complete with software and freebies, is an affordable alternative at only **\$179**.

**DataLink 1200.....\$179**

**DataLink 2400.....\$239**

## Order today!

To order or for more information, see your dealer or call (214) 241-6060 today, 9 am to 11 pm, 7 days. Or send check or money order to Applied Engineering. MasterCard, VISA and C.O.D. welcome. Texas residents add 7% sales tax. Add \$10 outside U.S.A.

**AE APPLIED ENGINEERING™**  
The Apple enhancement experts.

(214) 241-6060

P.O. Box 5100, Carrollton, TX 75011

4690-4800 start building a display string in what I call the STR.A00 buffer. It begins with token \$05, which means "position cursor". Then it inserts the line number in the format " d. " or "dd. ". Lines 4850-5000 insert the chosen cursor position after the \$05 token. Lines 5010-5140 append your string to this position and line number string. Finally, lines 5150-5190 display the result.

So you can see that to put a complete menu on the screen we would first zero LAST.LINE.NUMBER.OF.MENU; then clear the window we are using; then call DISPLAY.MENU.LINE once for each menu item; and finally, add any titles and explanations.

Once a menu is on the screen, the next logical step is to call SELECT.MENU.LINE so the user can tell you his choice. This subroutine is shown beginning at line 2000. You call it with a line number in the A-register. The subroutine even expects that you have just loaded that number into the A-register, because it starts with a BEQ instruction. If the EQ status is set, probably meaning you just did a LDA #0, the menu will begin with the bar on the first menu line. You will also get the first item if the value in A is larger than the last menu item number, or if A=1. You can begin with any other line by putting that line number in A. You could keep track of the item selected that last time this menu was used, like Copy II Plus does.

The subroutine finally returns when you type either ESCAPE, Apple-?, or RETURN. In the first two cases MENU.LINE.SELECTED and the A-register will be zero; in the third case they will be, you guessed it, the number of the menu line selected.

Lines 2000-2110 figure out which line you want to begin with, and inverse it. Unless, that is, the value in Z.A4 is non-zero. I think that variable must be an "escape flag" of some sort. If it is non-zero, SELECT.MENU.LINE exits the same way it would if you typed the ESCAPE key.

Lines 2120-2130 display the generic instructions for making a menu selection on the screen. The text of this message is shown in lines 5200-5230.

The rest of SELECT.MENU.LINE divides nicely into two main sections. Lines 2150-2770 handle the selection until you type a digit; lines 2780-3430 take over when you type a digit, and don't let go until you either type an arrow (up-, down, or left-), DELETE, RETURN, or ESCAPE. I noticed a significant chunk of duplicate code in the two sections: see lines 2420-2470 and lines 3330-3380. I feel certain that the code could be re-arranged a little and save space. We don't really need it, but a little here and a little there could make room for many new features.

Lines 2150-2190 invert the current line, move the cursor down to the bottom line at the end of the "Type..." message, and wait for you to type something. If you remember AW.KEYIN from the Feb 88 issue, it will also take its own action if you type certain keys. For example, Apple-/ gets changed into Apple-?;

Apple-Q and Apple-S store a non-zero value in Z.A4 and change the character to ESCAPE; and several other interesting things. Look at page 13 in that issue, lines 2580-2790, for a summary.

Look ahead to lines 2900-3000 for an interesting use of the Z.A4 flag. If you typed Apple-Q or Apple-S, it will be changed into ESCAPE by AW.KEYIN and Z.A4 will be set non-zero. Then the test at 2970-2980 will make the jump all the way to SML.ESCAPED. If you typed ESCAPE, that test will merely send you back to SML.WAITING. Interesting.... So if you are at this level, where the escape info line says "Escape: Erase entry": typing one ESCAPE takes you back to SML.WAITING; typing two ESCAPES would get you to SML.ESCAPED, or typing one Apple-Q would do the same thing. Whoever called SELECT.MENU.LINE can tell the difference, though, because Z.A4 will still be non-zero for Apple-Q or zero for two ESCAPES. (For some reason I never can remember what Apple-Q and -S are really used for.)

Lines 2240-2310 branch appropriately for up- or down-arrow or RETURN. Lines 2590-2770 handle the up- and down-arrows: simply a matter of changing the display of the current line back to normal mode, and raising or lowering the selected line number. The line number changes circularly, so that up-arrow off the top wraps around to the bottom, and down-arrow at the bottom goes to the top.

Lines 2340-2500 check for a digit and operate on it. Notice that only digits 1-9 are accepted here, while digits 0-9 are accepted at lines 3170-3200. This will be the first digit of a menu line number, so it cannot be zero. The other place it will be the second digit, so it could be zero. If this first digit is acceptable (between 1 and the last menu line number), lines 2420-2470 will echo the character at the cursor on line 23, restore the current line to normal mode, and store the new menu line number.

Lines 2480-2490 call a subroutine in the \$D000 area to change the escape and help info. This subroutine decides which pair of messages to display by the letter in the caller's A-register. Letter "E" makes escape say "Escape: Erase entry" and help say how much memory is left. Letter "S", used in lines 1910-1920, makes help say "Apple-? for help" and escape say whatever string was saved in a buffer at \$0CFD. The subroutine has several other options.

There appears to be either a bug or a leftover at lines 2520-2530. If you type Apple-? you get to this line. It loads up Z.89, and then acts like you typed ESCAPE. However, the contents of Z.89 are not needed here, because the first instruction after the JMP is another LDA!

I have probably already said enough about lines 2800-3430, especially since it is so similar to the section above. I would, however, like to look at a few lines. The code in lines 3210-3280 multiplies the previous menu line number by ten and adds in the next digit. At first glance it looks reasonably efficient for space, since it calls on a multiplication

subroutine. Nevertheless, I can recode it in one less byte and many fewer cycles without calling the subroutine. Change lines 3210-3280 to the following:

```

3210    AND #$0F          isolate new digit
3220    STA Z.91          save for later
3230    LDA MENU.LINE.SELECTED
3240    ASL               old * 2
3250    ASL               old * 4
3260    ADC MENU.LINE.SELECTED old*5
3270    ASL               old * 10
3280    ADC Z.91          plus new digit

```

Let's move ahead. When you type RETURN, lines 3600-3720 will clear line 23, restore the selected line to normal mode, and draw an arrow over the line number ("-->"). Either ESCAPE or RETURN brings you to lines 3740-3810, which erase the MENU.TABLE and return with the selected line number in the A-register. It makes no sense to me to erase MENU.TABLE, since it will not be used again until it is re-loaded anyway, but maybe there is some reason it is needed.

Lines 3860-4540 will copy the selected line off the screen into a buffer at \$0900, and then re-display it in one of three ways: the entire line NORMAL, the entire line INVERSE, or just the portion (A) characters long starting in column (X) in INVERSE.

Now if only AppleWorks could handle pathname entry as easily! The human interface presented by such Quit Code replacements as ProSel, Squirt, or the S-C Program Selector would be an improvement.

```

80-      1030 PSTR      .EQ $80,81
84-      1040 CHAR.FROM.KEYIN      .EQ $84
8C-      1050 MENU.LINE.SELECTED   .EQ $8C
89-      1060 Z.89      .EQ $89
91-      1070 Z.91      .EQ $91      result of multiply here
9A-      1080 P0        .EQ $9A      parameters from GET.4.PARMS
9B-      1090 P1        .EQ $9B      "
9C-      1100 P2        .EQ $9C      "
A4-      1110 Z.A4      .EQ $A4      escape flag???
1120    #-----
0A00-    1130 STR.A00    .EQ $A00
0FBE-    1140 MENU.CORNER.LEFT      .EQ $0FBE
0FBF-    1150 MENU.CORNER.TOP      .EQ $0FBF
1160    #-----
1170      .DUMMY
1180      .OR $0EB6
0EB6-    1190 MENU.TABLE.BS 3*31      Room for 30 3-byte entries
0F13-    1200 LAST.LINE.NUMBER.OF.MENU .BS 1
1210      .ED
1220    #-----Covered in AAL---
14D1-    1230 DISPLAY.STRING      .EQ $14D1      Jan 88, page 10
179D-    1240 CONVERT.A.TO.RJBF.STRING .EQ $179D
1818-    1250 BEEP.AND.CLEAR.KEYBUF   .EQ $1818
1823-    1260 MOVE.CURSOR.TO.XY      .EQ $1823      Feb 88, page 17
1B3A-    1270 MULTIPLY.X.BY.Y        .EQ $1B3A
187A-    1280 COPY.SCRN.LINE.TO.0900 .EQ $187A      Feb 88, page 18
18AE-    1290 GET.4.PARMS            .EQ $18AE      Dec 87, page 7
1D35-    1300 AW.KEYIN              .EQ $1D35      Feb 88, page 12
1E80-    1310 MOVE.CURSOR.TO.TCOL.TROW .EQ $1E80
1EA9-    1320 POINT.PSTR.AT.STR.A00   .EQ $1EA9      Feb 88, page 18
1EF8-    1330 MOVE.STRING            .EQ $1EF8      Dec 87, page 8
1F3E-    1340 DISPLAY.AT            .EQ $1F3E
1FE0-    1350 CLEAR.KEYBUF          .EQ $1FE0      Feb 88, page 17
2093-    1360 DISPLAY.STRING.PO      .EQ $2093
1FE9-    1370 DISPLAY.TOKEN.X        .EQ $1FE9
1FF5-    1380 DISPLAY.ON.LINE.23     .EQ $1FF5
1390    #-----
D023-    1400 SHOW.ESCAPE.AND.HELP.MSGS .EQ $D023
1410    #-----

```



### **SPECIAL !!! EXPANDED RAM/ROM BOARD: \$39.00**

Similar to our \$30 RAM/ROM dev board described below. Except this board has two sockets to hold your choice of 2-2K RAM, 2-2K ROM or even 2-4K ROM for a total of 8K. Mix RAM and ROM too. Although Apple limits access to only 2K at a time, soft switches provide convenient socket selection. Hard switches control defaults.

### **IMPROVED !!! II IN A MAC (ver 2.0): \$75.00**

Now includes faster graphics, UniDisk support and more! Bi-directional data transfers are a snap! This Apple II emulator runs DOS 3.3/PRODOS (including 6502 machine language routines) on a 512K MAC or MACPLUS. All Apple II features are supported such as HI/LO-RES graphics, 40/80 column text, language card and joystick. Also included: clock, RAM disk, keyboard buffer, on-screen HELP, access to the desk accessories and support for 4 logical disk drives. Includes 2 MAC diskettes (with emulation, communications and utility software, plus DOS 3.3 and PRODOS system masters, including Applesoft and Integer BASIC) and 1 Apple II diskette.

### **SCREEN.GEN: \$35.00**

Develop HI-RES screens for the Apple II on a Macintosh. Use MACPAINT (or any other application) on the MAC to create your Apple II screen. Then use SCREEN.GEN to transfer directly from the MAC to an Apple II (with SuperSerial card) or IIC. Includes Apple II diskette with transfer software plus fully commented SOURCE code.

### **MIDI-MAGIC for Apple //c: \$49.00**

Compatible with any MIDI equipped music keyboard, synthesizer, organ or piano. Package includes a MIDI-out cable (plugs directly into modem port - no modifications required!) and 6-song demo diskette. Large selection of digitized QRS player-piano music available for 19.00 per diskette (write for catalog). MIDI-MAGIC compatible with Apple II family using Passport MIDI card (or our own input/output card w/drum sync for only \$99.00).

### **FONT DOWNLOADER & EDITOR: \$39.00**

Turn your printer into a custom typesetter. Downloaded characters remain active while printer is powered. Use with any Word Processor program capable of sending ESC and control codes to printer. Switch back and forth easily between standard and custom fonts. Special functions (like expanded, compressed etc.) supported. Includes HIRES screen editor to create custom fonts and special graphics symbols. For Apple II, II+, //e. Specify printer: Apple Imagewriter, Apple Dot Matrix, C.Itoh 8510A (Prowriter), Epson FX 80/85, or Okidata 92/192.

\* **FONT LIBRARY DISKETTE #1: \$19.00** contains lots of user-contributed fonts for all printers supported by the Font Downloader & Editor. Specify printer with order.

### **DISASM 2.2e : \$30.00 (\$50.00 with SOURCE Code)**

Use this intelligent disassembler to investigate the inner workings of Apple II machine language programs. DISASM converts machine code into meaningful, symbolic source compatible with S-C, LISA, ToolKit and other assemblers. Handles data tables, displaced object code & even provides label substitution. Address-based triple cross reference generator included. DISASM is an invaluable machine language learning aid to both novice & expert alike. Don Lancaster says DISASM is "absolutely essential" in his ASSEMBLY COOKBOOK.

### **The 'PERFORMER' CARD: \$39.00 (\$59.00 with SOURCE Code)**

Converts a 'dumb' parallel printer I/F card into a 'smart' one. Simple command menu. Features include perforation skip, auto page numbering with date & title, large HIRES graphics & text screen dumps. Specify printer: MX-80 with Grafrax-80, MX-100, MX-80/100 with Grafraxplus, NEC 8092A, C.Itoh 8510 (Prowriter), Okidata 82A/83A with Okigraph & Okidata 92/93.

### **'MIRROR' ROM: \$25.00 (\$45.00 with SOURCE Code)**

Communications ROM plugs directly into Novation's Apple-Cat Modem card. Basic modes: Dumb Terminal, Remote Console & Programmable Modem. Features include: selectable pulse or tone dialing, true dialtone detection, audible ring detect, ring-back, printer buffer, 80 col card & shift key mod support.

### **RAM/ROM DEVELOPMENT BOARD: \$30.00**

Plugs into any Apple slot. Holds one user-supplied 2Kx8 memory chip (6116 type RAM for program development or 2716 EPROM to keep your favorite routines on-line). Maps into \$Cn00-CnFF and \$C800-CFFF.

### **C-PRINT For The APPLE //c: \$69.00**

Connect standard parallel printers to an Apple //c serial port. Separate P/S included. Just plug in and print!

-----  
Unless otherwise specified, all Apple II diskettes are standard (not copy protected!) 3.3 DOS.

Avoid a \$3.00 handling charge by enclosing full payment with order. VISA/MC and COD phone orders OK.

**RAK-WARE 41 Ralph Road W. Orange NJ 07052 (201) 325-1885**



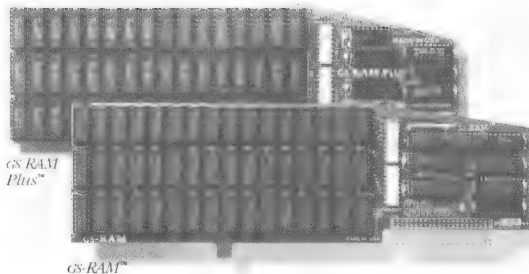
```

1420      .MA MSG      Macro to make a string with
1430      .DA #1-#-1  first byte is the length,
1440      .AS "j1"    the rest is ASCII.
1450      :1
1460      .EM
1470      *-----
1480      .PH $18DD
1490      *-----
1500      * Load X-reg with pointer into MENU.TABLE
1510      * which is menu line number times 3.
1520      * (18DD) 18E4 18F2 1A4D
1530      GET.MENU.TABLE.INDEX
18DD- A5 8C      LDA MENU.LINE.SELECTED
18DF- 0A      ASL
18E0- 65 8C      ADC MENU.LINE.SELECTED
18E2- AA      TAX
18E3- 60      RTS
1590      *-----
1600      * Re-display entire line in NORMAL mode.
1610      * (18E4) 1973 198B 199A 1A21 1A39 1A4A
1620      MAKE.MENU.LINE.NORMAL
18E4- 20 DD 18      JSR GET.MENU.TABLE.INDEX
18E7- BD B7 0E      LDA MENU.TABLE+1,X  Get screen line #
18EA- A8      TAY
18EB- A9 00      LDA #$00      Display entire line NORMAL
18ED- AA      TAX
18EE- 20 77 1A      JSR REVERSE.A.SCREEN.LINE
18F1- 60      RTS
1700      *-----
1710      * Re-Display the menu line in INVERSE mode.
1720      * (18F2) 1936 19AF 1A27
1730      MAKE.MENU.LINE.INVERSE
18F2- 20 DD 18      JSR GET.MENU.TABLE.INDEX
18F5- BD B8 0E      LDA MENU.TABLE+2,X  get length of menu msg
18F8- 48      PHA
18F9- BD B7 0E      LDA MENU.TABLE+1,X  get screen line #
18FC- A8      TAY
18FD- BD B6 0E      LDA MENU.TABLE,X    get starting column of msg
1900- 18      CLC
1901- 69 04      ADC #4      skip over "##. "
1903- AA      TAX      start at blank before msg itself
1904- 68      PLA      get length again
1905- 18      CLC
1906- 69 02      ADC #2      include leading and trailing blank
1908- 20 77 1A      JSR REVERSE.A.SCREEN.LINE
190B- 60      RTS
1880      *-----
1890      * (190C) 19D8 19E2 19EC 19F6
1900      RESTORE.ESCAPE.AND.HELP.MSGS
190C- A9 53      LDA #$53
190E- 20 23 D0      JSR SHOW.ESCAPE.AND.HELP.MSGS
1911- 20 80 1E      JSR MOVE.CURSOR.TO.TCOL.TROW
1914- 20 93 20      JSR DISPLAY.STRING.PO
1917- 1A 19      .DA TWO.BLANKS
1919- 60      RTS
1970      *-----
191A- 02 20 20      TWO.BLANKS .HS 02.20.20
1990      *-----
2000      * (A) = menu line number; if =0, or >lastline, use 1
2010      *-----
2020      SELECT.MENU.LINE
191D- F0 07      BEQ .1      Use top menu line
191F- CD 13 0F      CMP LAST.LINE.NUMBER.OF.MENU
1922- 90 04      BCC .2      Use selected line
1924- F0 02      BEQ .2      Use selected line (bottom)
1926- A9 01      LDA #1      Use top menu line
1928- 85 8C      STA MENU.LINE.SELECTED
192A- A5 A4      LDA Z.A4
192C- F0 03      BEQ .3
192E- 4C 3F 1A      JMP SML.ESCAPED.WITHOUT.INVERSE
1931- 20 35 1F      JSR DISPLAY.ON.LINE.23
1934- 6A 14      .DA MENU.MSG "Type number, or use arrows, then press Retu
2140      * (1936) 1997 19AC 19DB
2150      SML.WAITING
1936- 20 F2 18      JSR MAKE.MENU.LINE.INVERSE
1939- 20 E0 1F      .1 JSR CLEAR.KEYBUF
193C- 20 80 1E      JSR MOVE.CURSOR.TO.TCOL.TROW
193F- 20 35 1D      JSR AW.KEYIN
1942- C9 1B      CMP #$1B      did he type ESCAPE?
1944- D0 03      BNE .2      ...not ESCAPE
1946- 4C 32 1A      JMP SML.ESCAPED

```

# For Those Who Want the Most. From Those Who Make the Best. GS-RAM™

Now expand the IIgs' RAM and ROM with up to 8 MEG of "Instant On" memory with the all new GS-RAM!



GS-RAM has an all new design. A design that delivers higher performance including increased speed, greater expandability, and improved software.

## More Sophisticated, Yet Easier to Use

GS-RAM works with all IIgs software. In fact any program that runs on Apple's smaller memory card runs on the GS-RAM. But with GS-RAM you can have more memory, improved performance, and almost unlimited expansion capabilities. We've designed the new GS-RAM to be easier to use too—you don't have to adjust the size of your RAM disk every time you use a DMA device. No other RAM card with more than 4 banks of memory installed can make the same claim.

## More than Just Hardware

Each GS-RAM and GS-RAM Plus includes the most powerful set of IIgs software enhancements available anywhere. In fact, our nearest competitor offers only a fraction of the invaluable programs that we include with each GS-RAM card. This software includes the most powerful disk-caching program available, the GS-RAM Cache. The Cache will make most of your applications run up to 7 times faster. Also included is a diagnostic utility that lets you test your GS-RAM by graphically showing the location of any bad or improperly installed RAM chips. And for AppleWorks users, we give you our exclusive Expander program that dramatically enhances both the capabilities and speed of AppleWorks.

## Making AppleWorks Even Better

Applied Engineering's Expander program eliminates AppleWorks internal memory limits allowing it to recognize up to 8 megabytes of desktop workspace. You can increase the limits from only 7,250 lines to 22,600 lines in the word processor and from 6,350 records to 22,600 records in the database. The Expander allows all of AppleWorks, including print functions, to automatically load into RAM. The clipboard size will increase from 255 to 2,042 lines maximum. GS-RAM will automatically segment larger files so you can save them onto multiple floppies. And

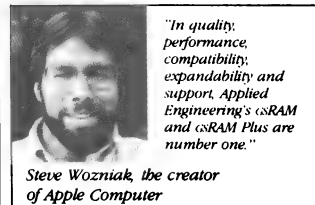
GS-RAM provides a built-in print buffer that allows you to continue working in AppleWorks while your printer is still processing text. You can even load Pinpoint or MacroWorks and your favorite spelling checker into RAM for instant response.

## Grow by Kilobytes or Megabytes

We offer GS-RAM in two configurations so you can increase your memory 256K at a time (GS-RAM) or a megabyte at a time (GS-RAM Plus). Both are IIgs compatible and both come with our powerful enhancement software. GS-RAM can hold up to 1.5 MEG of 256K chips and GS-RAM Plus can hold up to 6 MEG using 1 MEG chips. And since both use standard RAM chips (not high-priced SIMM's), you'll find expanding your GS-RAM or GS-RAM Plus easy, convenient, and very economical. For further expansion, you can plug a 2 MEG "piggyback" card into the GS-RAM's expansion port for up to 3.5 MEG of total capacity. Or up to a whopping 8 MEG on GS-RAM Plus. If a GS-RAM owner outgrows 3.5 MEG, he can easily upgrade to GS-RAM Plus for a nominal charge.

## Permanent Storage for an "Instant On" Apple

With our RamKeeper™ back-up option, your GS-RAM or GS-RAM Plus will retain both programs and data while your IIgs is turned off! Now when you turn your IIgs back on, your favorite software is on your screen in under 4 seconds! With RamKeeper you can divide your IIgs memory into part "electronic hard disk" and part extended RAM. Even change the memory boundaries at any time—and in any way—you want. Because



"In quality, performance, compatibility, expandability and support, Applied Engineering's GS-RAM and GS-RAM Plus are number one."

Steve Wozniak, the creator of Apple Computer

Applied Engineering has the most experience in the industry with battery-backed memory for the Apple, you are assured of the most reliable memory back-up system available. And in the world of battery-backed memory, *Reliability* is everything. That's why Applied Engineering uses state-of-the-art "GEL-CELL's" instead of Ni-Cad batteries (if Ni-Cads aren't discharged periodically, they lose much of their capacity). RamKeeper has about 6 hours of "total power failure" back-up time. That's 6 times the amount of other systems. But with power from your wall outlet, RamKeeper will back-up GS-RAM, GS-RAM Plus, or most other IIgs memory cards indefinitely. Should you ever have a "total power failure," RamKeeper switches to its 6-hour battery. When power returns, RamKeeper will automatically recharge the battery to full power. RamKeeper incorporates a dual-rate charger, status L.E.D.'s, and advanced power reducing circuitry. RamKeeper comes complete with battery, software, and documentation.

## GS-RAM's Got it ALL!

- 5-year warranty — parts & labor
- 6 RAM banks (most cards have 4)
- Memory expansion port
- ROM expansion port
- Ultra-fast disk caching on ProDOS 8 AND ProDOS 16.
- Expands AppleWorks internal limits
- Includes hi-res self test
- No soldered-in RAM chips
- Expandable to 8 MEG
- No configuration blocks to set
- RamKeeper back-up option allows permanent storage of programs & data
- 15-day money-back guarantee
- Proudly made in the USA.

GS-RAM with 256K	\$189
GS-RAM with 512K	\$259
GS-RAM with 1 MEG	\$399
GS-RAM with 1.5 MEG	\$539
GS-RAM with 2.5 to 3.5 MEG	CALL
GS-RAM Plus with 1-8 MEG	CALL
RamKeeper Option	\$179

## Order today!

See your dealer or call Applied Engineering today, 9 a.m. to 11 p.m. 7 days. Or send check or money order to Applied Engineering, MasterCard, VISA and C.O.D. welcome. Texas residents add 7% sales tax. Add \$10.00 outside USA.

**AE APPLIED ENGINEERING™**  
The Apple enhancement experts.

(214) 241-6060

P.O. Box 798, Carrollton, TX 75006

Prices subject to change without notice.

GS-RAM, GS-RAM Plus and RamKeeper are trademarks of Applied Engineering. Other brands and product names are registered trademarks of their respective holders.

```

2230 *-----
1949- C9 0D 2240 .2 CMP #$0D is it RETURN?
194B- D0 03 2250 BNE .3 ...not RETURN
194D- 4C 45 1A 2260 JMP SML.RETURN
2270 *-----
1950- C9 0B 2280 .3 CMP #$0B up arrow?
1952- F0 37 2290 BEQ SML.UP.ARROW
1954- C9 0A 2300 CMP #$0A down arrow?
1956- F0 42 2310 BEQ SML.DOWN.ARROW
1958- C9 BF 2320 CMP #$7F Apple and "?"
195A- F0 25 2330 BEQ .5 ...question mark
195C- C9 31 2340 CMP #$31
195E- 90 26 2350 BCC .6 ...not digit, beep!
1960- C9 3A 2360 CMP #$3A
1962- B0 22 2370 BCS .6 ...not digit, beep!
1964- 29 0F 2380 AND #$0F
1966- CD 13 0F 2390 CMP LAST.LINE.NUMBER.OF.MENU
1969- F0 02 2400 BEQ .4 ...bottom line
196B- B0 19 2410 BCS .6 ...beyond the bottom, beep!
196D- 48 2420 .4 PHA save the digit typed
196E- A6 84 2430 LDX CHAR.FROM.KEYIN
1970- 20 E9 1F 2440 JSR DISPLAY.TOKEN.X
1973- 20 E4 18 2450 JSR MAKE.MENU.LINE.NORMAL
1976- 68 2460 PLA get digit typed again
1977- 85 8C 2470 STA MENU.LINE.SELECTED select line 1-9
1979- A9 45 2480 LDA #$45
197B- 20 23 D0 2490 JSR SHOW.ESCAPE.AND.HELP.MSGS
197E- 4C AF 19 2500 JMP SML.DIGIT
2510 *---Apple-? typed-----
1981- A5 89 2520 .5 LDA Z.89 ...why? opcode JMPed to is LDA also!
1983- 4C 3F 1A 2530 JMP SML.ESCAPED.WITHOUT.INVERSE
2540 *-----
1986- 20 18 18 2550 .6 JSR BEEP.AND.CLEAR.KEYBUF
1989- D0 AE 2560 BNE .1 ...always
2570 *-----
2580 * (198B) 1952 19EF
2590 SML.UP.ARROW
198B- 20 E4 18 2600 JSR MAKE.MENU.LINE.NORMAL
198E- C6 8C 2610 DEC MENU.LINE.SELECTED
1990- D0 05 2620 BNE .1 ...still in range
1992- AD 13 0F 2630 LDA LAST.LINE.NUMBER.OF.MENU
1995- 85 8C 2640 STA MENU.LINE.SELECTED
1997- 4C 36 19 2650 .1 JMP SML.WAITING
2660 *-----
2670 * (199A) 1956 19F9
2680 SML.DOWN.ARROW
199A- 20 E4 18 2690 JSR MAKE.MENU.LINE.NORMAL
199D- A5 8C 2700 LDA MENU.LINE.SELECTED
199F- CD 13 0F 2710 CMP LAST.LINE.NUMBER.OF.MENU
19A2- B0 04 2720 BCS .1 ...already on bottom, wrap to top
19A4- E6 8C 2730 INC MENU.LINE.SELECTED next line down
19A6- D0 04 2740 BNE .2 ...always
19A8- A9 01 2750 .1 LDA #1 top line now
19AA- 85 8C 2760 STA MENU.LINE.SELECTED
19AC- 4C 36 19 2770 .2 JMP SML.WAITING
2780 *-----
2790 * (19AF) 197E
2800 SML.DIGIT
19AF- 20 F2 18 2810 JSR MAKE.MENU.LINE.INVERSE
19B2- 20 80 1E 2820 .1 JSR MOVE.CURSOR.TO.TCOL.TROW
19B5- A2 07 2830 LDX #$07 CURSOR RIGHT TOKEN
19B7- 20 E9 1F 2840 JSR DISPLAY.TOKEN.X
19BA- A5 8C 2850 LDA MENU.LINE.SELECTED
19BC- C9 0A 2860 CMP #10 on line 1-9?
19BE- 90 05 2870 BCC .2 ...yes
19C0- A2 07 2880 LDX #$07 CURSOR RIGHT TOKEN
19C2- 20 E9 1F 2890 JSR DISPLAY.TOKEN.X
19C5- 20 35 1D 2900 .2 JSR AW.KEYIN get another char, maybe 2nd digit
19C8- C9 1B 2910 CMP #$1B is it ESCAPE?
19CA- F0 08 2920 BEQ .3 ...yes
19CC- C9 08 2930 CMP #$08
19CE- F0 04 2940 BEQ .3 ...backspace
19D0- C9 7F 2950 CMP #$7F ...is it DELETE?
19D2- D0 0A 2960 BNE .4 ...yes
19D4- A5 A4 2970 .3 LDA Z.A4
19D6- D0 5A 2980 BNE SML.ESCAPED
19D8- 20 0C 19 2990 JSR RESTORE.ESCAPE.AND.HELP.MSGS
19DB- 4C 36 19 3000 JMP SML.WAITING

```

```

19DE- C9 0D 3010 *-----
19E0- D0 06 3020 .4 CMP #$0D is it RETURN?
19E2- 20 0C 19 3030 BNE .5 ...no
19E5- 4C 45 1A 3040 JSR RESTORE.ESCAPE.AND.HELP.MSGS
3050 JMP SML.RETURN
3060 *-----
19E8- C9 0B 3070 .5 CMP #$0B up arrow?
19EA- D0 06 3080 BNE .6 ...no
19EC- 20 0C 19 3090 JSR RESTORE.ESCAPE.AND.HELP.MSGS
19EF- 4C 8B 19 3100 JMP SML.UP.ARROW
3110 *-----
19F2- C9 0A 3120 .6 CMP #$0A down arrow?
19F4- D0 06 3130 BNE .7 ...no
19F6- 20 0C 19 3140 JSR RESTORE.ESCAPE.AND.HELP.MSGS
19F9- 4C 9A 19 3150 JMP SML.DOWN.ARROW
3160 *-----
19FC- C9 30 3170 .7 CMP #$30 another digit?
19FE- 90 2D 3180 BCC .10 ...not a digit, beep!
1A00- C9 3A 3190 CMP #$3A
1A02- B0 29 3200 BCS .10 ...not a digit, beep!
1A04- 48 3210 PHA save the digit
1A05- A6 8C 3220 LDX MENU.LINE.SELECTED
1A07- A0 0A 3230 LDY #10 multiply previous line# by 10
1A09- 20 3A 1B 3240 JSR MULTIPLY.X.BY.Y
1A0C- 68 3250 PLA add the new digit
1A0D- 29 0F 3260 AND #$0F
1A0F- 18 3270 CLC
1A10- 65 91 3280 ADC Z.91 product here
1A12- B0 19 3290 BCS .10 ...too large, beep!
1A14- CD 13 0F 3300 CMP LAST.LINE.NUMBER.OF.MENU
1A17- F0 02 3310 BEQ .8 ...bottom line
1A19- B0 12 3320 BCS .10 ...too large, beep!
1A1B- 48 3330 PHA save the new line number
1A1C- A6 84 3340 LDX CHAR.FROM.KEYIN
1A1E- 20 E9 1F 3350 JSR DISPLAY.TOKEN.X
1A21- 20 E4 18 3360 JSR MAKE.MENU.LINE.NORMAL
1A24- 68 3370 PLA get the new line number
1A25- 85 8C 3380 STA MENU.LINE.SELECTED
1A27- 20 F2 18 3390 JSR MAKE.MENU.LINE.INVERSE
1A2A- 4C B2 19 3400 JMP .1
3410 *-----
1A2D- 20 18 18 3420 .10 JSR BEEP.AND.CLEAR.KEYBUF
1A30- D0 F8 3430 BNE .9 ...always
3440 *-----
3450 * (1A32) 1946 19D6
3460 SML.ESCAPED
1A32- A5 A4 3470 LDA Z.A4
1A34- 48 3480 PHA
1A35- A9 00 3490 LDA #$00
1A37- 85 A4 3500 STA Z.A4
1A39- 20 E4 18 3510 JSR MAKE.MENU.LINE.NORMAL
1A3C- 68 3520 PLA
1A3D- 85 A4 3530 STA Z.A4
3540 * (1A3F) 192E 1983
3550 SML.ESCAPED.WITHOUT.INVERSE
1A3F- A9 00 3560 LDA #$00
1A41- 85 8C 3570 STA MENU.LINE.SELECTED
1A43- F0 1B 3580 BEQ CLEAR.MENU.TABLE.AND.EXIT ...ALWAYS
3590 *-----
3600 * (1A45) 194D 19E5
3610 SML.RETURN
1A45- 20 F5 1F 3620 JSR DISPLAY.ON.LINE.23
1A48- 72 1A 3630 .DA I.1A72 1 Byte String, Clear-Line Token
1A4A- 20 E4 18 3640 JSR MAKE.MENU.LINE.NORMAL
1A4D- 20 DD 18 3650 JSR GET.MENU.TABLE.INDEX
1A50- BD B7 0E 3660 LDA MENU.TABLE+1,X
1A53- A8 3670 TAY draw an arrow "-->" over number
1A54- BD B6 0E 3680 LDA MENU.TABLE,X
1A57- AA 3690 TAX
1A58- 20 23 18 3700 JSR MOVE.CURSOR.TO.XY
1A5B- 20 93 20 3710 JSR DISPLAY.STRING.PO
1A5E- 6D 1A 3720 .DA ARROW
3730 * (1A60) 1A43
3740 CLEAR.MENU.TABLE.AND.EXIT
1A60- A9 00 3750 LDA #0
1A62- A2 5E 3760 LDX #3*31+1 entire table & highest line number
1A64- 9D B5 0E 3770 .1 STA MENU.TABLE-1,X
1A67- CA 3780 DEX
1A68- D0 FA 3790 BNE .1
1A6A- A5 8C 3800 LDA MENU.LINE.SELECTED
1A6C- 60 3810 RTS

```

```

3820 *-----
1A6D- 3830 ARROW >MSG " -->"
1A72- 01 02 3840 I.1A72 .HS 01.02 1 BYTE STRING, CLEAR LINE TOKEN
3850 *
3860 * Re-display a screen line in INVERSE or NORMAL mode
3870 * (Y) = screen line to be re-displayed
3880 * (X) = column number to begin with
3890 * (A) = 0 then re-display entire line NORMAL
3900 * (A) > 78 then redisplay entire line INVERSE
3910 * (A) = length of middle section of line to be
3920 * displayed in INVERSE.
3930 *-----
1A74- 3940 RASL.X .BS 1
1A75- 3950 RASL.Y .BS 1
1A76- 3960 RASL.A .BS 1
3970 *-----
3980 * (1A77) 103F 18EE 1908
3990 REVERSE.A.SCREEN.LINE
1A77- 8E 74 1A 4000 STX RASL.X Save the column
1A7A- 8C 75 1A 4010 STY RASL.Y Save the line
1A7D- 8D 76 1A 4020 STA RASL.A Save the length
1A80- 98 4030 TYA get line number
1A81- 20 7A 18 4040 JSR COPY.SCRN.LINE.TO.0900
1A84- A2 00 4050 LDX #0 start in column 0
1A86- AC 75 1A 4060 LDY RASL.Y on line Y
1A89- 20 23 18 4070 JSR MOVE.CURSOR.TO.XY
1A8C- AD 76 1A 4080 LDA RASL.A get specified length
1A8F- F0 09 4090 BEQ .1 ...0 means display it all NORMAL
1A91- C9 4F 4100 CMP #79
1A93- 90 1C 4110 BCC .2 ...it really is a length
1A95- A2 0A 4120 LDX #$0A ...>78, INVERSE the line
1A97- 20 E9 1F 4130 JSR DISPLAY.TOKEN.X
1A9A- AD FA 1A 4140 .1 LDA HANDLE.0900
1A9D- 85 80 4150 STA PSTR
1A9F- AD FB 1A 4160 LDA HANDLE.0900+1
1AA2- 85 81 4170 STA PSTR+1
1AA4- A9 4F 4180 LDA #79 display 79 characters
1AA6- 20 D1 14 4190 JSR DISPLAY.STRING
1AA9- A2 0B 4200 LDX #$0B NORMAL TOKEN
1AAB- 20 E9 1F 4210 JSR DISPLAY.TOKEN.X
1AAE- 4C F9 1A 4220 JMP .3 ...ONLY AN RTS
4230 *-----
1AB1- AD FA 1A 4240 .2 LDA HANDLE.0900 Point to line image
1AB4- 85 80 4250 STA PSTR
1AB6- AD FB 1A 4260 LDA HANDLE.0900+1
1AB9- 85 81 4270 STA PSTR+1
1ABB- AD 74 1A 4280 LDA RASL.X Display up to X in NORMAL
1ABE- 20 D1 14 4290 JSR DISPLAY.STRING
1AC1- A2 0A 4300 LDX #$0A INVERSE TOKEN
1AC3- 20 E9 1F 4310 JSR DISPLAY.TOKEN.X
1AC6- AD 74 1A 4320 LDA RASL.X Display middle section INVERSE
1AC9- 85 80 4330 STA PSTR
1ACB- AD FB 1A 4340 LDA HANDLE.0900+1
1ACE- 85 81 4350 STA PSTR+1
1AD0- AD 76 1A 4360 LDA RASL.A # chars in middle section
1AD3- 20 D1 14 4370 JSR DISPLAY.STRING
1AD6- A2 0B 4380 LDX #$0B NORMAL TOKEN
1AD8- 20 E9 1F 4390 JSR DISPLAY.TOKEN.X
1ADB- AD 74 1A 4400 LDA RASL.X Display rest of line NORMAL
1ADE- 18 4410 CLC
1ADF- 6D 76 1A 4420 ADC RASL.A
1AE2- B0 15 4430 BCS .3 ...line too long, forget the rest
1AE4- C9 4F 4440 CMP #79
1AE6- B0 11 4450 BCS .3 ...line too long, forget the rest
1AE8- 85 80 4460 STA PSTR
1AEA- AD FB 1A 4470 LDA HANDLE.0900+1
1AED- 85 81 4480 STA PSTR+1
1AEF- A9 4F 4490 LDA #79 79 chars
1AF1- 38 4500 SEC
1AF2- E5 80 4510 SBC PSTR
1AF4- F0 03 4520 BEQ .3 ...no space left on the line
1AF6- 20 D1 14 4530 JSR DISPLAY.STRING
1AF9- 60 4540 RTS
4550 *-----
1AFA- 00 09 4560 HANDLE.0900 .DA $0900

```

```

4570 *-----
4580 .PH $2029
4590 *-----
4600 *   Display string P2 at P0,P1 after "xx. "
4610 *   where xx is decimal form of (LAST.LINE.NUMBER.OF.MENU)
4620 *   LAST.LINE.NUMBER.OF.MENU is incremented by this routine
4630 *   Stores 3 bytes in MENU.TABLE:
4640 *   column, line, and string length
4650 *   MENU.TABLE has room for 31 such entries.
4660 *-----
4670 DISPLAY.MENU.LINE
2029- 20 AE 18 4680 JSR GET.4.PARMS      Get P0-P3
202C- EE 13 OF 4690 INC LAST.LINE.NUMBER.OF.MENU
202F- A9 05 4700 LDA #$05             "GO TO XY" TOKEN
2031- 8D 00 0A 4710 STA STR.A00
2034- AD 13 OF 4720 LDA LAST.LINE.NUMBER.OF.MENU
2037- 20 9D 17 4730 JSR CONVERT.A.TO.RJBF.STRING
203A- 8E 03 0A 4740 STX STR.A00+3        blank or first digit of ##
203D- 8C 04 0A 4750 STY STR.A00+4        units digit of ##
2040- A9 2E 4760 LDA #'.'
2042- 8D 05 0A 4770 STA STR.A00+5        string is "##. "
2045- A9 20 4780 LDA #'.'
2047- 8D 06 0A 4790 STA STR.A00+6
204A- 8D 07 0A 4800 STA STR.A00+7
204D- AD 13 OF 4810 LDA LAST.LINE.NUMBER.OF.MENU
2050- 0A 4820 ASL                multiply by 3
2051- 6D 13 OF 4830 ADC LAST.LINE.NUMBER.OF.MENU
2054- AA 4840 TAX                save for index into MENU.TABLE
4850 *---Column number-----
2055- A5 9A 4860 LDA P0          Get specified column
2057- 10 06 4870 BPL .1          ...it is absolute
2059- 29 7F 4880 AND #$7F        ...it is relative
205B- 18 4890 CLC
205C- 6D BE OF 4900 ADC MENU.CORNER.LEFT
205F- 8D 01 0A 4910 .1 STA STR.A00+1 Put column # into string
2062- 9D B6 OE 4920 STA MENU.TABLE,X and save in table
4930 *---Line number-----
2065- A5 9B 4940 LDA P1          Get specified line number
2067- 10 06 4950 BPL .2          ...it is absolute
2069- 29 7F 4960 AND #$7F        ...it is relative
206B- 18 4970 CLC
206C- 6D BF OF 4980 ADC MENU.CORNER.TOP
206F- 8D 02 0A 4990 .2 STA STR.A00+2 Put line # into string
2072- 9D B7 OE 5000 STA MENU.TABLE+1,X and save in table
5010 *---Append text of menu line-----
2075- A0 00 5020 LDY #0
2077- B1 9C 5030 LDA (P2),Y      Save length of string in table
2079- 9D B8 OE 5040 STA MENU.TABLE+2,X
207C- A8 5050 TAY          Point at last char in string
207D- 18 5060 CLC          Compute length of combined strings
207E- 69 08 5070 ADC #8          05.xx.yy "##. " = 8 chars
2080- AA 5080 TAX          Point at last char of combination
2081- 48 5090 PHA          Save total length for display subr.
2082- B1 9C 5100 .3 LDA (P2),Y get next char of caller's string
2084- 9D FF 09 5110 STA STR.A00-1,X append to ours
2087- CA 5120 DEX
2088- 88 5130 DEY
2089- D0 F7 5140 BNE .3          ...more to copy
5150 *---Display the string-----
208B- 20 A9 1E 5160 JSR POINT.PSTR.AT.STR.A00
208E- 68 5170 PLA          Get length off stack
208F- 20 D1 14 5180 JSR DISPLAY.STRING
2092- 60 5190 RTS
5200 *-----
5210 .PH $146A
146A- 5220 MENU.MSG >MSG "Type number, or use arrows, then press Return "
5230 *-----

```

# In about the time it takes to read this headline, you can have the Finder up and



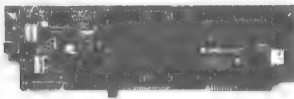
**N**ow your favorite program can be ready to go seconds after you flip your Apple IIcs on.

With Applied Engineering's RamKeeper™ card, your IIcs retains stored programs *and* data when you turn your computer off.

RamKeeper powers up to two memory cards simultaneously when your Apple IIcs is off. And battery backup keeps power to the boards even during power failures. Your programs and data are stored in a permanent, "electronic hard disk," always ready to run.

#### **Superior power backup.**

Applied Engineering has the most experience in battery-



*RamKeeper lets you keep programs and data in permanent, "electronic hard disk" memory. Turn your Apple IIcs on and you're ready to work.*

backed memory for Apple computers. We were the first to offer battery-backed memory with our RamFactor™/RamCharger™ combination. Now RamKeeper sets the standard for IIcs memory backup.

Our experience shows in the way we designed and built RamKeeper. We used sealed Gel/

Cell batteries — far more reliable than Ni-Cads in this application. Ni-Cads lose much of their capacity if they're not discharged periodically. Just when you need them most, Ni-Cads could run out of power.

Our Gel/Cell pack, which is included in our price, gives you up to six hours of total power failure backup. That's about 6 times longer than other systems.

RamKeeper uses a Switching Power Supply — the same technology used by Apple for the IIcs power supply. This design uses energy much more efficiently to keep your Apple running cooler.

Our sealed Gel/Cell battery

stays *outside* your computer case. With other systems, the batteries are installed under the IIgs power supply where a leak could ruin computer circuitry.

### Put two memory boards in the same slot.

You might have bought your IIgs with Apple's memory card. But now you want the features of Applied's *gs-RAM™* card. RamKeeper efficiently resolves the dilemma.

You can use RamKeeper with your current IIgs memory card *and* add another memory card — all in the same slot. Just attach your current memory card to one side of your new RamKeeper card, connect the second card to the other side and plug RamKeeper into the slot.

Of course RamKeeper works fine with just one memory card. But you can use two and still keep Slot 7 open with our optional Slot-Mover.

### Makes all of your memory usable memory.

RamKeeper can power up to 16 Meg of memory. Other systems are limited to only 8 Meg. In addition, RamKeeper lets you mix and match different types of cards. For example, you can have a *gs-RAM Plus™* using 1 Meg RAM chips and an Apple card using 256K RAM chips. Other systems are limited in the combinations they allow.

RamKeeper firmware automatically configures for two cards



*It all comes with RamKeeper ... board, Gel/Cell battery pack, easy-to-understand instructions, and Applied's powerful AppleWorks Expander software.*

when the second card is installed. Other systems make you manually move jumpers.

RamKeeper configures memory linearly. Other systems don't, so they create memory gaps that can cause program crashes or keep some programs from using as much as half of your memory.

You easily decide how much memory you'll devote to ROM and to RAM from the IIgs Desk Accessories menu. You can configure Kilobytes or Megabytes of instant ROM storage for your favorite programs. And you can change ROM size any time without affecting stored files.

### Protected from program crashes.

RamKeeper controlling firmware is in an EPROM. A program crash can't take out the operating software. With other systems, operating software is installed in RAM from a floppy. If the program crashes, it can take the operating software with it, and reinstalling the disk-based operating software destroys data in memory.

### Verifies data security.

RamKeeper firmware uses optional startup checksums to verify that no data has been lost while power was off. The firmware also

runs ROM and RAM memory tests without disturbing data on the card.

### Free AppleWorks Enhancement software.

Applied's powerful AppleWorks Enhancement software is free with RamKeeper. It makes AppleWorks faster and far more powerful by eliminating AppleWorks internal memory limits. Word processor limits go from only 7,250 lines to 22,600 lines. Database limits go from 6,350 records to 22,600 records. The clipboard size limit is increased from 255 to 2,042 lines. It even automatically segments large files so you can save them on multiple floppies. No other company expands your IIgs' AppleWorks internal limits.

In addition, the most powerful disk-caching program available comes with the RamKeeper. The cache significantly increases access time to the Apple 3.5 Drive. Most applications will run up to 7 times faster.

### The largest maker of Apple expansion boards.

Applied Engineering has sold more expansion boards than anyone else. And we've been in business 8 years, long enough to see the vast majority of our competitors come and go.

All of our products are crafted in the U.S.A. We back RamKeeper with a five year parts and labor warranty. And a 15-day, no questions asked, money back guarantee.

### Only \$189.00.

See your dealer. Or call 214-241-6060, 9 a.m. to 11 p.m. 7 days. Or send check or money order to Applied Engineering, MasterCard, VISA, C.O.D. welcome. Texas residents add 7% sales tax. Add \$10.00 outside U.S.A. Prices subject to change without notice.

**AE APPLIED ENGINEERING™**  
**The Apple enhancement experts.**  
 P.O. Box 5100 • Carrollton, Texas 75011.  
 (214) 241-6060



*RamKeeper is easy to install. Just plug it in. Even when you use two memory boards, you don't have jumpers. You can have two memory boards but use only one slot.*

RamKeeper, Ramfactor, *gs-RAM*, and *gs-RAM Plus* are trademarks of Applied Engineering.

Have you ever wanted to convert a file from one filetype to another? I have, and it seems that there should be a command in BASIC.SYSTEM to allow it. Some other command shells, such as Davex by DAL Systems, do have such a command.

Of course, the flexibility of the BLOAD and BSAVE commands does allow you to get the job done. You can BLOAD the source file, specifying the filetype and load address; CREATE an empty file of the new type; and BSAVE into that new file, specifying the filetype, beginning RAM address, and length.

But what if you want to go just a little further, and also make some slight changes to the file's contents? Then you need an intelligent file transformer. You need a program that will ask for an input and an output pathname, read the input file and transform the data appropriately, and write the transformed data on the output file. This article presents a specific file transformer, and you may either use it as is or modify it to your own needs.

I have often been asked, by programmers who did not own the S-C Macro Assembler, for a program which would convert S-C source code files into ordinary text files. Back in the December, 1983, issue of Apple Assembly Line I published such a program, but it was written in Applesoft for running under DOS 3.3. I don't believe that program would operate properly under ProDOS, but even if it did it would be terribly slow.

Under DOS 3.3, S-C source code is stored in type "I" files. DOS does all the actual LOADING and SAVEing, being fooled into believing that the source code is Integer BASIC. Under ProDOS, S-C source code is stored in type \$FA files, which Apple has set aside for Integer BASIC source programs. Since Integer BASIC has never been supported under ProDOS, and probably never will be, I chose to use this file type for S-C source code. In fact, when you are using the S-C Macro Assembler this filetype is called "S-C".

The source code text is stored in a slightly compressed format in these files. The same format is used for both the DOS and ProDOS versions, so that you can use Apple's CONVERT program or a utility like COPY II PLUS to move files from one operating system to the other. If you were to read a file byte-by-byte under both operating systems, you would notice one difference: under DOS the file's length is stored in the first two bytes; under ProDOS the length is kept in the directory.

Each line of S-C source code begins with a line number. This number may be any value from 0 to 65535. The transformer should write the source lines on the output text file without this line number, because most assemblers using text files for source do not use explicit line numbers.

Each line of S-C source code is stored in the file in a form first defined by Steve Wozniak's Integer BASIC:

```
byte 1:  number of bytes in entire line (n)
byte 2:  lo-byte of line number in binary
byte 3:  hi-byte of line number in binary
bytes 4...n-1:  tokenized form of source line
byte n:  00
```

For example, an empty line with line number 1000 would be stored as:

```
04 E8 03 00
```

The tokenized form of S-C source is almost plain ASCII. Each byte is stored in ASCII with bit 7 zero, with two exceptions. First, blanks are stored in a compressed form. One blank is stored with the code \$81; two blanks are stored with the single code \$82; in general, a string of n consecutive blanks is stored with the single code \$80+n. The largest compressed blank code is \$BF, which stands for 63 consecutive blanks. If there are more than 63 blanks in a row, they will be represented by two or more compressed-blank codes. For example, the line

```
1010 STAR   LDA #0       Zero
```

would be stored as:

```
14          number of bytes in line = 20
F2 03       line number is $03F2 = 1010
53 54 41 52 "STAR"
83          three blanks
4C 44 41 81 "LDA "
23 30 85    "#0  "
5A 65 72 6F "Zero"
00          end of line token
```

While blank is the most frequently repeated character in a source program, there are others. The token \$C0, followed by a repetition count and an ASCII character, represents any string of the same character. For example, the line

```
1020 *-----
```

would be stored as:

```
08          8 bytes in line
FC 03       line number 1020
2A          "*"
C0 20 2D    means 32 consecutive "--"
00          end of line
```

With the above information, you can see that the file transformer will have to read in the first byte of each line, which tells how many bytes are in the rest of that line. Then it should read in the rest of that line. Once the entire line is in RAM, the transformer should scan through the bytes of the

line, copying ASCII characters to the new line, and expanding any compressed characters into the new line. Finally, the new line should be written on the output file. When the transformer reaches the end of the input file it is finished.

Let me insert here a reminder that even though this transformer is very specific, you should be able to easily modify it to handle other types of transformations. For example, the subscriber data base I use to produce mailing labels every month is kept on a series of plain text files, understandable only to the mailing list program I wrote about eight years ago. Even though it is plain text, AppleWorks cannot read it into a structured database unless I make some "transformations" first. At a minimum, I need a transformer that will split the city, state, and zip code line into three separate lines. I could easily write such a transformer by modifying the following program a little. Another transformer could read a binary file and write out a text file in the Intel or Motorola hex format, for later transmission through a serial port to an EPROM programmer. The possibilities are endless.

Now back to the program ALREADY written. I wrote two versions, and you can assemble either one of them by changing line 1060. As shown here, the fancier version is selected. The simpler version assumes specific filenames assembled into the code in lines 4340-4480. The fancier version prompts for filenames or pathnames to be typed in when you run the program. The code for both versions is listed, but only the fancy version was actually assembled.

Lines 1200-1400 define four macros I used in the program. The first one is for calling the ProDOS Machine Language Interface, or MLI. The second one creates a call to a subroutine which prints 00-terminated strings, followed by the string to be printed. The third generates a call to a subroutine which prints strings which begin with a byte count. The fourth generates calls to a subroutine which reads a line from the keyboard into a specified buffer, and then stores the length in the first byte of the buffer.

It turns out I do not use both the PRSTR and RDSTR macros, just one or the other. In the simple version, PRSTR is used to display the pre-assembled filenames; in the fancy version, RDSTR is used to let you type in the filenames.

Lines 1450-1550 call on two subroutines to open the input and output files. If they are successful, the file reference numbers returned by MLI will be stored into parameter blocks for reading and writing those files. If not, the transformer program will just close all files and end. I'll discuss the two opening subroutines later.

I decided after a few tests that I wanted some sort of progress indication on the screen while the program was busy. I decided to list the line number on the screen. Line 1570 uses the PRINT macro to display "LINE NUMBER: ". Inside the main loop, at line 1680, I call DISPLAY.LINE.NUMBER to display the current line number in five-digit decimal, and then backspace 5 times.

This makes a very attractive (to me) indicator. In a more general transformer, you might change this to display a hexadecimal file position, or some other meaningful parameter.

The main loop runs from line 1590 to line 1790. First lines 1590-1620 try to read three bytes. If there is any error, I assume it is due to reaching the end of the source file, and that ends the loop. If no error, then the input buffer contains the byte count for the whole line, and the line number. Since we already read the first three bytes, lines 1630-1670 compute the number of bytes left in the line and set up the parameter block to read that many. Then I print out the line number as described above. The DISPLAY.LINE.NUMBER subroutine also checks to see if any key has been pressed on the keyboard, which is interpreted to mean you want to abort the transformer. Line 1690 branches in that case, and the files are closed. Lines 1700-1710 read the rest of the source line.

Line 1730 calls the CONVERT.ONE.LINE subroutine, which scans the source line and builds an expanded output line. Lines 1740-1780 write the expanded line on the output file. Any error returned by MLI from this write ends the main loop. Probably I should have printed out an error message for this condition, because it is abnormal. It could happen if you were trying to write into a write-protected file or on a write-protected disk. Looking at it now, I think I would change line 1780 to "BCS .4", and insert the following lines:

```
1862 .4      JSR OPEN.ERROR
1864         >PRINT "\UNABLE TO WRITE ON OUTPUT FILE"
1866         RTS
```

Once the main loop ends, lines 1800-1820 truncate the output file. It may be that your output file already existed, and was longer than necessary to contain the new information. These lines chop off any extra data. Line 1810 reads the current file position, and line 1820 sets that value as the new end-of-file. Finally, line 1840 closes both files.

I ended the program with a simple RTS. You might want a fancier ending. For example, you might want a message on the screen saying "I AM FINISHED" or the like. Then you might want a short menu on the screen allowing a choice of transforming another file, ending with an RTS, or doing a ProDOS QUIT call. See, I left something for you to do!

If you do start adding features, you might also like to add the ability to transform a range of lines from the source file, by specifying a beginning and ending line number. You might want to add tests for the correct file types on the input and output files. You might want to add a menu-style pathname entry, so that you could work with complicated directory structures without a photographic memory.

That last suggestion leads me to warn that the program given here requires that you enter complete pathnames, or else have a legitimate prefix set. It does not allow slot and drive specification using ",Sx" or ",Dx" either. Now don't let me

discourage you by listing all the things I left out. As is, the program is quite useful.

Lines 1870-1970 are the subroutine to open the input file. It first prints the prompt message "INPUT: ", and then waits for you type in a pathname. (If you selected the "simple" version, it instead prints out the pre-assembled pathname.) Line 1950 calls on ProDOS to open the file. If there is any error, lines 1990-2050 print out the error number and the subroutine returns with carry set. If there is no error, the subroutine returns with carry clear. You might want to add some code to this subroutine to read the filetype and make sure it is type \$FA.

Lines 2070-2270 open the output file. This is a little more complicated, because the output file may or may not already exist. If the file does not yet exist, the open call will return with error number \$46. Line 2160 tests for this error number. Any other error will be printed out, and the subroutine will return with carry set. Lines 2180-2260 attempt to CREATE a file. First the current date and time are read and stored in the parameter block, and then ProDOS is called to create the file. The parameter block used here assumes you want the output file to be type TXT (\$04). If you modify the program for a different transformation you may want to change the file type. You might also want to add some code to check an existing file for the correct file type.

Lines 2290-2510 are the parameter blocks for the various MLI calls I used.

Lines 2560-2830 display the current line number and test for a keypress. The subroutine converts the binary line number to decimal one digit at a time. Nothing spectacular, and in fact it is very similar to the subroutine Woz used inside Integer BASIC over ten years ago. Lines 2740-2780 send out five backspaces to put the cursor back over the first digit. I originally tried to do this by just storing a cursor position back in location \$24, but that only works in 40-column mode. Using backspaces it also works in 80-column mode, even with the 80-column cards used in Apple II+ machines. Lines 2790-2830 check for a keypress and return carry set if there was one.

Finally to the heart of this transformer: lines 2870-3330 convert one input line into one output line. Lines 2950-2970 zero two pointers, one for the input line scan and the other for the output line fill. These are used by the GET.CHAR and PUT.CHAR subroutines in lines 3180-3330. GET.CHAR picks up the character pointed to from the input line, and advances the pointer. PUT.CHAR stores the character in the A-register into the output line where the pointer points, and then advances the pointer. PUT.CHAR.MULTIPLE uses the X-register to store multiple copies of the character in the A-register.

Lines 2980-3000 pick up the next character from the input line, and branch according to its type. If the character is 00, this is the end of the line: then lines 3150 tack an ASCII <RETURN> code on the end of the output line, and the subroutine is finished. If the character is positive, it is simple ASCII and

lines 3010 simply copy it to the output line. If the character is negative, it is either a compressed blank string or a compressed string of some other character. Lines 3030-3090 handle compressed blanks by putting the blank count into the X-register, a blank in the A-register, and calling PUT.CHAR.MULTIPLE. Lines 3100-3130 handle the other kind of repeated character by reading the repeat count and character code from the input line and then calling PUT.CHAR.MULTIPLE.

Lines 3340-3550 are very similar in function to the PRINT subroutine I published last month in my SHOW INDEX program. A difference is the use of the "\" character in the string. Last month's program would have merely printed the "\". The program this month will print a <RETURN> when it sees a "\". This enabled me to use the PRINT macro to generate calls to the PRINT subroutine.

Lines 3560-3830 are a subroutine for printing strings which begin with a length byte. This subroutine is not ever called in the fancy version of my program given here. In the simple version it would be called to print the pre-assembled pathnames. I went ahead and assembled the subroutine anyway, because it is an interesting one. The macro PRSTR will call it, putting the address of the string to be printed as data immediately after the JSR PRSTR.

The subroutine used in the fancy version to read in a pathname is given in lines 3870-4250. The address of the buffer for receiving the pathname is given as data following the JSR RDSTR. Lines 3880-4010 accomplish the task of copying this address into the code below, into lines 4150 and 4240. Where the listing shows \$3333, the address of the buffer will be stored. The loop in lines 4030-4210 keeps reading characters, storing them into that buffer, and displaying them on the screen.

If a backspace is read, the character at the end of the buffer is backed out and the string backspace-space-backspace is displayed on the screen. When a <RETURN> is read, the loop terminates and the number of bytes before the <RETURN> is stored in the first byte of the buffer by lines 4200-4250.

The subroutine GET.VIA.PNTR in lines 4270-4320 is called by both PRSTR and RDSTR to pickup the address which follows the JSR calling those subroutines.

Lines 4340-4480 either assemble two 65-byte buffers (fancy version) or two pathnames. Lines 4490 to the end assemble the other buffers used. Since the buffers used by ProDOS for the open files must begin on a page boundary, line 4530 skips ahead to the next page boundary.

For those of you who do not own the S-C Macro Assembler, I will start including this file transformer in executable form on future issues of the AAL Monthly Disk. Remember, you can save yourself a lot of typing by subscribing to the Monthly Disk. These disks include all of the source code in S-C format and also the text of all of the articles.

```

1000 *SAVE SC.2.TEXT
1020 ***line missing above was ".LIST MOFF,CON"
1030 *-----
1040 .OR $2000
1050 *-----
01- 1060 FANCY .EQ 1 =0 to use pre-assembled file names
1070 *-----
FDOC- 1080 RDKEY .EQ $FDOC A few handy Monitor subroutines
FD8E- 1090 CROUT .EQ $FD8E
FDDA- 1100 PRBYTE .EQ $FDDA
FDED- 1110 COUT .EQ $FDED
1120 *-----
BF90- 1130 MLI.DATE .EQ $BF90 thru BF93 System DATE and TIME
1140 *-----
C000- 1150 KEYBOARD .EQ $C000 For aborting a conversion run
C010- 1160 STROBE .EQ $C010
1170 *-----
00- 1180 PIN .EQ $00 Scanning pointer for input line
01- 1190 POUT .EQ $01 Stuffing pointer for output line
1200 *-----
1210 .MA MLI
1220 JSR $BF00
1230 .DA #$11,12
1240 .EM
1250 *-----
1260 .MA PRINT
1270 JSR PRINT Print 00-term'd string after
1280 .AS -"1" here is the string
1290 .HS 00 here is the 00-terminator
1300 .EM
1310 *-----
1320 .MA PRSTR
1330 JSR PRSTR Print string beginning with byte count
1340 .DA 11 address of string
1350 .EM
1360 *-----
1370 .MA RDSTR Read a string from keyboard or EXEC
1380 JSR READ.STRING
1390 .DA 11 address of string
1400 .EM
1410 *-----
1420 * Program to read an S-C Macro source file
1430 * and write it as a text file.
1440 *-----
1450 FILE.TRANSFORMER
2000- 20 78 20 1460 JSR OPEN.INPUT.FILE Open Source File
2003- B0 69 1470 BCS .3 ...unable to open it
2005- AD F4 20 1480 LDA IREF Get RefNum for READ
2008- 8D FC 20 1490 STA IOB.READ+1
1500 *---Open the text file-----
2008- 20 A8 20 1510 JSR OPEN.OUTPUT.FILE
200E- B0 5E 1520 BCS .3 ...unable to open it
2010- AD FA 20 1530 LDA OREF Get RefNum for WRITE
2013- 8D 0C 21 1540 STA IOB.EOF+1 and for Truncation
2016- 8D 04 21 1550 STA IOB.WRITE+1
1560 *---Print "LINE NUMBER"-----
2019- 1570 >PRINT "LINE NUMBER: " for progress indicator
1580 *---read byte count from source file
202C- A9 03 1590 .1 LDA #3 read 3 bytes: byte count + line #
202E- 8D FF 20 1600 STA IOB.READ+4
2031- 1610 >MLI CA,IOB.READ
2037- B0 29 1620 BCS .2 END OF FILE
1630 *---read rest of line from source file
2039- 38 1640 SEC rest of line is 3 less
203A- AD D6 22 1650 LDA LINE.SC than byte count
203D- E9 03 1660 SBC #3
203F- 8D FF 20 1670 STA IOB.READ+4
2042- 20 12 21 1680 JSR DISPLAY.LINE.NUMBER to indicate progress
2045- B0 1B 1690 BCS .2 ...wants to abort
2047- 1700 >MLI CA,IOB.READ
204D- B0 13 1710 BCS .2 END OF FILE
1720 *---Build Output Line-----
204F- 20 5A 21 1730 JSR CONVERT.ONE.LINE
1740 *---write line on text file-----
2052- A5 01 1750 LDA POUT
2054- 8D 07 21 1760 STA IOB.WRITE+4 # BYTES TO WRITE
2057- 1770 >MLI CB,IOB.WRITE
205D- B0 03 1780 BCS .2
205F- 4C 2C 20 1790 JMP .1

```

```

1800 *---truncate text file-----
2062- 1810 .2 >MLI CF,IOB.EOF GET MARK
2068- 1820 >MLI DO,IOB.EOF SET EOF
1830 *---Close both files-----
206E- 1840 .3 >MLI CC,IOB.CLOSE
2074- 20 8E FD 1850 JSR CROUT
2077- 60 1860 RTS
1870 *-----
1880 OPEN.INPUT.FILE
2078- 1890 >PRINT "\ INPUT: "
1900 .DO FANCY
2086- 1910 >RDSTR PATHI
1920 .ELSE
1930 >PRSTR PATHI
1940 .FIN
208B- 1950 >MLI C8,IOB.OPENI
2091- B0 01 BCS OPEN.ERROR
2093- 60 1970 RTS
1980 *-----
1990 OPEN.ERROR
2094- 48 2000 PHA
2095- 2010 >PRINT "\ERROR: "
20A2- 68 2020 PLA
20A3- 20 DA FD 2030 JSR PRBYTE
20A6- 38 2040 SEC
20A7- 60 2050 RTS
2060 *-----
2070 OPEN.OUTPUT.FILE
20A8- 2080 >PRINT "\OUTPUT: "
2090 .DO FANCY
20B6- 2100 >RDSTR PATHO
2110 .ELSE
2120 >PRSTR PATHO
2130 .FIN
20BB- 2140 .1 >MLI C8,IOB.OPENO
20C1- 90 1F 2150 BCC .3
20C3- C9 46 2160 CMP #46 was it FILE NOT FOUND?
20C5- D0 CD 2170 BNE OPEN.ERROR
20C7- 2180 >MLI 82,0
20CD- A0 03 2190 LDY #3
20CF- B9 90 BF 2200 .2 LDA MLI.DATE,Y
20D2- 99 EB 20 2210 STA IOB.CREATE+8,Y
20D5- 88 2220 DEY
20D6- 10 F7 2230 BPL .2
20D8- 2240 >MLI C0,IOB.CREATE
20DE- 90 DB 2250 BCC .1
20E0- B0 B2 2260 BCS OPEN.ERROR
20E2- 60 2270 .3 RTS
2280 *-----
20E3- 07 2290 IOB.CREATE .HS 07
20E4- 95 22 DA PATHO
20E6- C3 04 00
20E9- 00 01 00
20EC- 00 00 00 2310 .HS C3.04.0000.01.0000.0000
2320 *-----
20EF- 03 2330 IOB.OPENI .HS 03
20F0- 54 22 2340 .DA PATHI
20F2- 00 25 2350 .DA BUF.I
20F4- 2360 IREF .BS 1
2370 *-----
20F5- 03 2380 IOB.OPENO .HS 03
20F6- 95 22 2390 .DA PATHO
20F8- 00 29 2400 .DA BUF.O
20FA- 2410 OREF .BS 1
2420 *-----
20FB- 04 00 2430 IOB.READ .HS 04.00
20FD- D6 22 2440 .DA LINE.SC
20FF- 00 00 2450 .DA 0
2101- 00 00 2460 .DA 0
2470 *-----
2103- 04 00 2480 IOB.WRITE .HS 04.00
2105- D6 23 2490 .DA LINE.TEXT
2107- 00 00 2500 .DA 0
2109- 00 00 2510 .DA 0
2520 *-----
210B- 02 00 00
210E- 00 00 2530 IOB.EOF .HS 02.00.00.00.00
2110- 01 00 2540 IOB.CLOSE .HS 01.00

```

```

2550 *-----
2560 DISPLAY.LINE.NUMBER
2570 LDY #4
2580 LDX #0
2590 .1 LDA LINE.SC+1
2600 CMP DECTBL,Y
2610 LDA LINE.SC+2
2620 SBC DECTBH,Y
2630 BCC .3
2640 INX
2650 STA LINE.SC+2
2660 LDA LINE.SC+1
2670 SBC DECTBL,Y
2680 STA LINE.SC+1
2690 JMP .2
2700 .3 TXA
2710 JSR COUT
2720 DEY
2730 BPL .1
2740 LDY #5
2750 LDA #88
2760 .4 JSR COUT
2770 DEY
2780 BNE .4
2790 LDA KEYBOARD
2800 CMP #80 SET CARRY IF ANY KEY
2810 BCC .5
2820 STA STROBE
2830 .5 RTS
2840 *-----

2150- 01 0A 64 2850 DECTBL .DA #1,#10,#100,#1000,#10000
2153- E8 10
2155- 00 00 00
2158- 03 27 2860 DECTBH .DA /1,/10,/100,/1000,/10000
2870 *-----
2880 *---build output line
2890 *--- ignore line numbers
2900 *--- expand blanks and multiples
2910 *--- use low ascii
2920 *--- end with 0D (return)
2930 *-----
2940 CONVERT.ONE.LINE
2950 LDY #0
2960 STY POUT
2970 STY PIN
2980 .1 JSR GET.CHAR
2990 BEQ .4 ...End of Line
3000 BMI .2 ...blanks or other multiple
3010 JSR PUT.CHAR simple ASCII char
3020 JMP .1
3030 .2 CMP #CO
3040 BCS .3 ...other multiple
3050 AND #7F
3060 TAX
3070 LDA #20
3080 JSR PUT.CHAR.MULTIPLE
3090 JMP .1
3100 .3 JSR GET.CHAR get count
3110 TAX
3120 JSR GET.CHAR get repeated char
3130 JSR PUT.CHAR.MULTIPLE
3140 JMP .1
3150 .4 LDA #OD
3160 JMP PUT.CHAR
3170 *-----
3180 GET.CHAR
3190 LDY PIN
3200 INC PIN
3210 LDA LINE.SC,Y
3220 RTS
3230 *-----
3240 PUT.CHAR
3250 LDX #1
3260 PUT.CHAR.MULTIPLE
3270 LDY POUT
3280 .1 STA LINE.TEXT,Y
3290 INY
3300 DEX
3310 BNE .1
3320 STY POUT
3330 RTS

```

```

3340 *-----
3350 PRINT
3360 PLA
3370 STA PNTR+1 POP RETURN ADDRESS
3380 PLA BECAUSE IT POINTS TO STRING
3390 STA PNTR+2
3400 JSR PRINT.VIA.PNTR
3410 LDA PNTR+2 PUT RETURN ADDRESS ON STACK
3420 PHA
3430 LDA PNTR+1
3440 PHA
3450 RTS
3460 *-----
3470 PVP ORA #$80
3480 CMP #"\
3490 BNE .1
3500 LDA #$8D
3510 .1 JSR COUT PRINT CHAR
3520 PRINT.VIA.PNTR
3530 JSR GET.VIA.PNTR GET NEXT CHAR OF STRING
3540 BNE PVP 00 = END OF STRING
3550 RTS
3560 *-----
3570 * Print a string which begins with a byte count
3580 * JSR PRSTR
3590 * .DA address of byte count
3600 *-----
3610 PRSTR
3620 PLA
3630 STA PNTR+1
3640 PLA
3650 STA PNTR+2
3660 JSR GET.VIA.PNTR
3670 TAX
3680 JSR GET.VIA.PNTR
3690 TAY
3700 LDA PNTR+2 PUT RETURN ADDRESS ON STACK
3710 PHA
3720 LDA PNTR+1
3730 PHA
3740 STX PNTR+1
3750 STY PNTR+2 POINT AT STRING NOW
3760 JSR PNTR GET BYTE COUNT
3770 TAX
3780 .1 JSR GET.VIA.PNTR GET NEXT CHAR OF STRING
3790 ORA #$80
3800 JSR COUT
3810 DEX
3820 BNE .1
3830 RTS

```

## DON LANCASTER STUFF

### INTRODUCTION TO POSTSCRIPT

A 65 min user group VHS video with Don Lancaster sharing many of his laser publishing and Postscript programming secrets.

Includes curve tracing, \$5 toner refilling, the full Kroy Kolor details, page layouts, plus bunches more.

**\$39.50**

### ASK THE GURU

An entire set of reprints to Don Lancaster's ASK THE GURU columns, all the way back to column one. Edited and updated.

Both Apple and desktop publishing resources are included that are not to be found elsewhere.

**\$24.50**

### APPLE IIc/IIe ABSOLUTE RESET

Now gain absolute control over your Apple! You stop any program at any time.

Eliminates all dropouts on your HIRES screen dumps. Gets rid of all hole blasting. For any IIc or IIe.

**\$19.50**

### POSTSCRIPT SHOW & TELL

Unique graphics and text routines the others don't even dream of. For most any Postscript printer.

Fully open, unlocked, and easily adaptable to your own needs. Available for Apple, PC, Mac, ST, many others.

**\$39.50**

FREE VOICE HELPLINE

VISA/MC

**SYNERGETICS**

Box 809-SC

Thatcher, AZ 85552

(602) 428-4073

```

3840 *-----
3850 *   Input a string from the keyboard (or EXEC file)
3860 *-----
3870 READ.STRING
21F7- 68      3880      PLA
21F8- 8D 51 22 3890      STA PNTR+1
21FB- 68      3900      PLA
21FC- 8D 52 22 3910      STA PNTR+2
21FF- 20 48 22 3920      JSR GET.VIA.PNTR
2202- 8D 34 22 3930      STA .2+1
2205- 8D 45 22 3940      STA .3+1
2208- 20 48 22 3950      JSR GET.VIA.PNTR
220B- 8D 35 22 3960      STA .2+2
220E- 8D 46 22 3970      STA .3+2
2211- AD 52 22 3980      LDA PNTR+2      PUT RETURN ADDRESS ON STACK
2214- 48      3990      PHA
2215- AD 51 22 4000      LDA PNTR+1
2218- 48      4010      PHA
4020 *-----
2219- A2 01      4030      LDX #1
221B- 20 0C FD 4040      JSR RDKEY
221E- 09 80      4050      ORA #$80
2220- C9 88      4060      CMP #$88
2222- D0 0F      4070      BNE .2
2224- CA      4080      DEX
2225- F0 F2      4090      BEQ .0
2227- 20 ED FD 4100      JSR COUNT
222A- A9 A0      4110      LDA #" "
222C- 20 ED FD 4120      JSR COUNT
222F- A9 88      4130      LDA #$88
2231- D0 08      4140      BNE .25
2233- 9D 33 33 4150      STA $3333,X
2236- E0 40      4160      CPX #64
2238- B0 E1      4170      BCS .1
223A- E8      4180      INX
223B- 20 ED FD 4190      JSR COUNT
223E- C9 8D      4200      CMP #$8D
2240- D0 D9      4210      BNE .1
2242- CA      4220      DEX
2243- CA      4230      DEX
2244- 8E 33 33 4240      STX $3333
2247- 60      4250      RTS
4260 *-----
4270 GET.VIA.PNTR
2248- EE 51 22 4280      INC PNTR+1      BUMP POINTER TO NEXT CHAR
224B- D0 03      4290      BNE PNTR
224D- EE 51 22 4300      INC PNTR+1
2250- AD 33 33 4310      LDA $3333      GET NEXT CHAR OF STRING
2253- 60      4320      RTS
4330 *-----
4340      .DO FANCY
2254- 4350      PATHI .BS 65
4360      .ELSE
4370      PATHI .DA #PATHI.LEN
4380      .AS "SC.2.TEXT"
4390      PATHI.LEN .EQ *-PATHI-1
4400      .FIN
4410 *-----
4420      .DO FANCY
2295- 4430      PATHO .BS 65
4440      .ELSE
4450      PATHO .DA #PATHO.LEN
4460      .AS "TTT"
4470      PATHO.LEN .EQ *-PATHO-1
4480      .FIN
4490 *-----
22D6- 4500      LINE.SC .BS 256
23D6- 4510      LINE.TEXT .BS 256
4520 *-----
24D6- 4530      .BS *-255/256*256-*      Pad to next page
2500- 4540      BUF.I .BS $400
2900- 4550      BUF.O .BS $400
4560 *-----

```

Modify CATALOG to Show All AuxTypes.....Bob Sander-Cederlof

Apple Computer has started spreading the word that they are going to make further use of the AuxType field in the ProDOS file directory. Until now it has had three primary purposes, depending on the file type.

Binary files (type BIN, or \$06) use the AuxType field to hold the loading address. System files (type SYS, or \$FF) files do too, but since they always load at \$2000 it is never used. Applesoft files (type BAS, or \$FC) also use it for the loading address, but I don't believe it is ever used. I haven't tested it, though. If you use the S-C Macro Assembler under ProDOS, it uses a filetype called "S-C" for source code files. This is type \$FA, which ProDOS reserved for Integer BASIC, just in case. S-C used type I files under DOS, so I decided to use type INT files under ProDOS. Anyway, I put the loading address in AuxType for these files too. It is never used.

Text files use AuxType to specify the record length for random-access files; it is 0000 for sequential files.

AppleWorks files (types ADB or \$19, AWP or \$1A, and ASP or \$1B) are the most creative, using 15 of the 16 AuxType bits to modify the file name characters. Each bit corresponds to one of the file name characters, allowing AppleWorks to show and recognize the file name with lower-case letters and spaces in it. Regular ProDOS only displays upper-case letters in the names, and does not allow spaces. AppleWorks could have just stored the real ASCII codes in the filename field, but then the rest of ProDOS would not be able to access the files. By using the AuxType field, all is compatible.

Probably there are some other uses for AuxType that I do not know about. And now Apple has decided to use it to allow more than 256 different file types. It seems that 256 is not going to be enough now that we have ProDOS-16. If you are writing software for the Apple market and you need some private filetypes, you are supposed to write Apple Developer Tech Support and tell them of your needs. They in turn will assign you the appropriate file types. Since there are more requests than 256, they are starting to categorize them. For example, there may be a future file type for all word processors, with different AuxType values to distinguish various kinds of word processor files.

Apple has suggested that we start displaying the AuxType information in CATALOGs for more types than just TXT and BIN. I looked into BASIC.SYSTEM and found a patch that makes it display for all file types. I also looked into the S-C Macro Assembler's SCASM.SYSTEM for the same kind of patch.

In BASIC.SYSTEM, you need to change one byte from \$27 to \$11. The byte is at \$A513 after BASIC.SYSTEM is operating, so you could do a POKE 42259,17 to turn the feature on, and POKE 42259,39 to turn it off. If you want to make a permanent change, you can do it this way:

```

}BLOAD BASIC.SYSTEM,TSYS,A$2000
}POKE 12051,17
}BSAVE BASIC.SYSTEM,TSYS,A$2000

```

Location 12051 is \$2F13, which is where the code resides when the file is BLOADED as above.

In SCASM.SYSTEM, you need to change one byte from \$16 to \$00. The byte is near \$B000 when the S-C Macro Assembler is operating, but the exact location depends on the particular edition you have. I suggest dis-assembling starting at \$AFF0 and looking for the following code:

```

C9 04  CMP #$04
D0 04  BNE ...
A9 D2  LDA #$D2
D0 06  BNE ...
C9 06  CMP #$06
D0 16  BNE ...      change this "16" to "00"
A9 C1  LDA #$C1

```

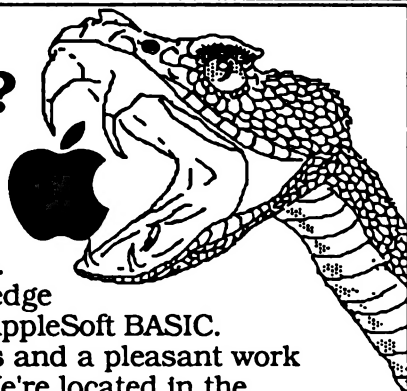
In the version I was using that "16" byte was at \$B00E. From within the assembler I typed "\$B00E:00" and that did the trick. If the "\$" commands don't work in your computer, you can go to the monitor by typing "MNT" and then make the patch.

If you want to make a permanent change, you need to BLOAD SCASM.SYSTEM, patch the byte, and BSAVE it again just like I did with BASIC.SYSTEM above. In my version, I found the byte at \$510E when the file was loaded.

## Do You Have Apple Knowledge?

If you do, Applied Engineering would like to put your knowledge to work. We're looking for someone to fill a position in our Technical Support group. You must have a strong working knowledge of AppleWorks, ProDOS, DOS 3.3, and AppleSoft BASIC. Applied Engineering offers good benefits and a pleasant work environment. Office is non-smoking. We're located in the Carrollton area.

So if you've got the knowledge and want to sink your teeth into a position with an ever-expanding company, give us a call at (214) 241-6060.



BLOADing Directories.....Bob Sander-Cederlof

Did you know that ProDOS will let you BLOAD a directory just like any other kind of file? I did not until today.

For example, if I want to load the directory of my Sider hard disk into memory, I can type BLOAD /HARD1,TDIR,A\$1000. I can load in any subdirectory the same way. This works under both BASIC.SYSTEM (Applesoft) and SCASM.SYSTEM (the S-C Macro Assembler) shells.

In both cases, if you care to, you can find the length of the directory in bytes in locations \$BEDB and \$BEDC. \$BEDB will always contain 00, and \$BEDC will be the number of pages in the directory, or twice the number of blocks.

I tried BSAVEing... but it is prohibited. You get a FILE LOCKED message for your efforts.

## EnterSoft:

Basic-like macros which make the complex simple. Don't re-write that multiplication routine for the hundredth time! Get EnterSoft instead! Do 8/16/32/64 bit Math/Input-Output/Graphics simply without all of the hassles. These routines are a must for the serious programmer who doesn't want to spend all of his/her time trying to re-invent the wheel. DOS 3.3 Version=\$30.00, ProDos Version=\$30.00, BOTH for only \$50.00. GET YOURS TODAY.

## A Shape Table Program:

For once! A shape table program which is logically organized into its componet parts. Each section resides in its own program. The editor, disk access, Hi-Res section; each section is separate. Written almost entirely in Basic, it is easily modified. Not copyprotected! Put them on a Hard Disk, Ram Drive, anywhere! DOS 3.3 Version=\$20.00, ProDos Version=\$20.00, BOTH for \$30.00!

Send Check or Money Order To:		ProDos Upgrade for DOS 3.3 EnterSoft Owners - \$20.00
c/o	Mark Manning Simulacron I/Baggy Game P.O. Box 58598 Webster, TX 77598	Thanks for the letters - Keep Writing!

Visiting Phoenix, continued from front page.

Bill Mensch gave several interesting talks regarding the history of the 6502 family and its future. The 65832 is on the horizon, but not fully defined. We know it will have 32-bit registers, multiply/divide instructions, and probably even floating point arithmetic. We can also be certain it will be plug compatible with the 65816, so we will be able to plop them directly into Apple IIgs sockets. We still have opportunity to send ideas for its features to Bill. Suggestion: do it in writing, be brief, keep any such letter to one or two ideas and no more than two pages. If you have more ideas, send them in separate letters, giving Bill time to digest them. Send your ideas directly to Bill at Western Design Center, 2166 East Brown Road, Mesa, AZ 85213. If you are VERY serious, give him a call at (602)962-4545.

You might also want to write for information about their new line of microCOMPUTERS: the W65C134 includes a 65C02, RAM, ROM, and oodles of I/O goodies all in one 68-pin package; the W65C265 will be even better, and built around a 65C816; and the W65C365 will include a 65C832.

A real highlight of the trip was meeting Don Lancaster, and his wife and daughter. Don was there to show and sell his many books and disks full of Apple and Postscript knowledge, and both he and wife Bea gave several seminars during the conference. If you are doing ANYTHING with laser printers, you need to look at what Don has. His "Ask the Guru" column in Computer Shopper is a gold mine, and you can get a neatly bound set of reprints for only \$24.50. Speaking of gold mines reminds me of caves, and the fact that the Lancasters are avid spelunkers.

Bill and I also enjoyed our lengthy discussions with Jeff Creamer, Steve Stephenson, and John & Ron Wrenholt. Jeff is a teacher in Prescott, AZ, and has published in these pages. Steve is chief programmer for Checkmate Technology, and gave me some wonderful disk files about AppleWorks. Expect to see some of his stuff here in the near future. John is the founder of Big Red Apple Club, now called Big Red Computer Club. He and brother Ron have quite a lot to offer any Print Shop aficionados, including a new program which lets you print labels with color graphics.

"Do not be conformed to this world, but be transformed  
by the renewing of your mind, that you may discover  
the good, and acceptable, and perfect will of God."

Apple Assembly Line (ISSN 0889-4302) is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is \$24 per year in the USA, Canada, and Mexico, or \$36 in other countries. Back issues are \$1.80 each for Volumes 1-7, \$2.40 each from later Volumes (plus postage). A subscription to the newsletter with a Monthly Disk containing all program source code and article text is \$64 per year in the USA, Canada and Mexico, and \$90 to other countries.

All material herein is copyrighted by S-C SOFTWARE CORPORATION, all rights reserved. (Apple is a registered trademark of Apple Computer, Inc.)

Page 32 .....Apple Assembly Line.....April, 1988.....Copyright (C) S-C SOFTWARE